

TADs y Programación Orientada a Objetos

Estructura de Datos/Algoritmos I

Depto. de computación-UNRC

Prof. Pablo Castro

Implementación de TADS

- Los TADS agrupan los datos con las operaciones permitidas sobre estos.
- Cuando implementamos un TAD queremos separar su especificación de su implementación.
- Los “clientes” del TAD no necesitan saber cómo está implementado.
- Esto permite intercambiar implementaciones de TADS; por ejemplo, pilas con listas, pilas con arreglos, etc.

Prog. Procedimental vs POO

Programación procedimental:

- El código se organiza alrededor de la idea de procedimiento.
- El mecanismo principal de abstracción es el de procedimiento.
- Funciona bien cuando los datos son simples, el mecanismo de abstracción sirve para organizar *funcionalidad y no datos*.
- La estructuración de datos que se provee es limitada: arreglos, registros, etc.

Programación orientada a objetos

- Provee mecanismos potentes para la abstracción de datos
- **Encapsulamiento**: las operaciones se asocian a los datos sobre los cuales estas actúan.
- **Ocultamiento**: Solo las operaciones asociadas a los datos pueden realizar cambios sobre ellos.
- Estas características mejoran el **reuso** y la **modularidad** del código.

POO

La programación orientada a objetos está basada en la idea de objeto:

Un objeto es una colección de datos y procedimientos.

La noción de objeto está vinculada al concepto de clase:

Una clase es una colección de objetos que comparten la misma estructura

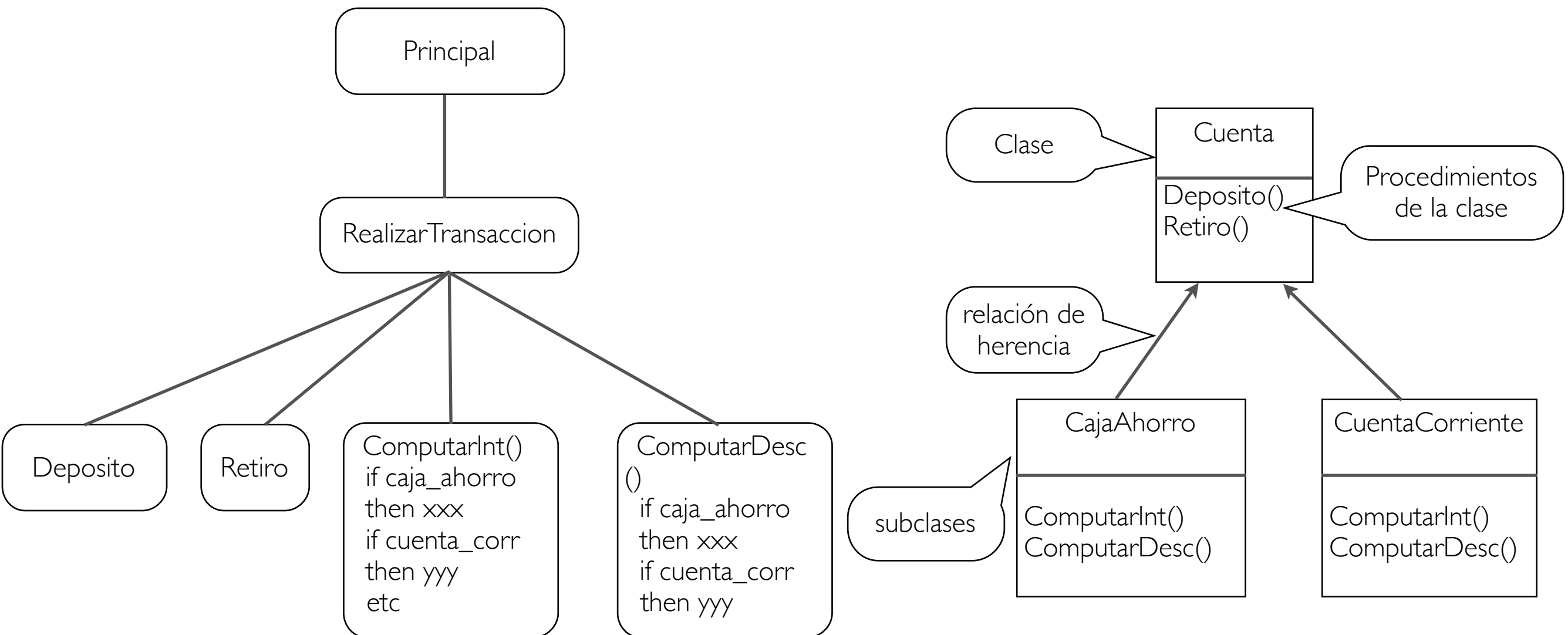
Cada objeto es la instancia de una clase

Las clases son abstracciones de datos, que incluyen abstracciones de procedimientos

La noción de clase es muy parecida a la noción de TAD

POO vs PP

Veamos un pequeño ejemplo para comparar ambos paradigmas: una cuenta de banco



POO

En los lenguajes POO que usaremos tenemos:

- Tipos básicos (enteros, char, strings, etc.)
- Tipos estructurados (arreglos, etc.)
- Punteros (o referencias).
- Sentencias imperativas de programación.
- Mecanismos avanzados de modularidad (clases).
- Mecanismos avanzados de reutilización. (Herencia y polimorfismo)

La noción de clase

Una clase es un mecanismo de modularización que encapsula:

- La definición de estructura de datos, por medio de variables de otro tipo (llamadas atributos).
- La definición de rutinas que permiten manipular estos datos.

Los lenguajes POO proveen diversos mecanismos para declarar la visibilidad de los datos o procedimientos.

Un ejemplo

Consideremos el siguiente ejemplo en pseudo-código.

clase Circulo

atributos

centro_x: real [privado]
centro_y: real [privado]
radio: real [privado]

metodos

centro() : (real,real) [publico]
 retornar (centro_x, centro_y)
fin
radio() : real [publico]
 retornar (radio)
fin
circunferencia() : real [publico]
 retornar (2*radio*pi)
fin
nuevo_centro(x,y: real) [publico]
 centro_x:=x; centro_y:=y
fin
nuevo_radio(r: real) [publico]
 radio:=r
fin
fin_clase

Estos son los atributos
(datos) de la clase

visibilidad de los
datos

visibilidad de las
rutinas

Estos son los métodos
(rutinas) de la clase

Clases como tipos

Cada clase define un nuevo tipo, en el ejemplo anterior tendríamos un nuevo

Dos variables de tipo
Circulo (objetos)

```
...  
c1,c2: Circulo
```

se crean los objetos (se
crea su espacio en
memoria)

```
...  
c1.nuevo_centro(1,5);  
c1.nuevo_radio(10);
```

```
...  
Si c2.radio() = 0 entonces c2.nuevo_radio(c1.radio())
```

```
...  
// esto no puede hacerse  
c2.radio:=0;  
c2.nuevo_radio(c1.radio);
```

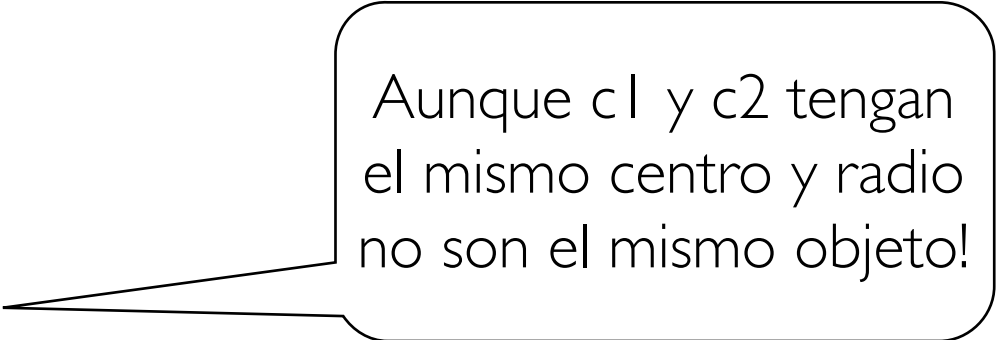
El punto nos permite llamar a
rutinas/datos de los objetos

No puede hacerse
porque se viola la
visibilidad de los datos

Instancias

- Se pueden tener múltiples instancias de una clase.
- Cada instancia es un objeto.
- Cada objeto tiene identidad propia, más allá de los valores de sus atributos

```
...  
c1,c2: Circulo  
...  
c1.nuevo_centro(1,5);  
c1.nuevo_radio(10);  
c2.nuevo_centro(1,5);  
c2.nuevo_radio(10);  
// c1 y c2 son objetos distintos!
```



Aunque c1 y c2 tengan
el mismo centro y radio
no son el mismo objeto!

Constructores y destructores

- La inicialización de un objeto se realiza en su constructor.
- En muchos lenguajes OO es necesario decir como se destruye el objeto (por ejemplo: C++ pero no java).

```
clase Circulo
```

```
    Circulo() [publico]
```

```
        centro_x: 0;
```

```
        centro_y: 0;
```

```
        radio: 0
```

```
fin
```

```
~Circulo() [publico]
```

```
    ...
```

```
fin
```

constructor: se dice
cuales son los valores del
objeto al inicio

Se dice cómo se
destruye el objeto. En
JAVA no hace falta!

Constructores

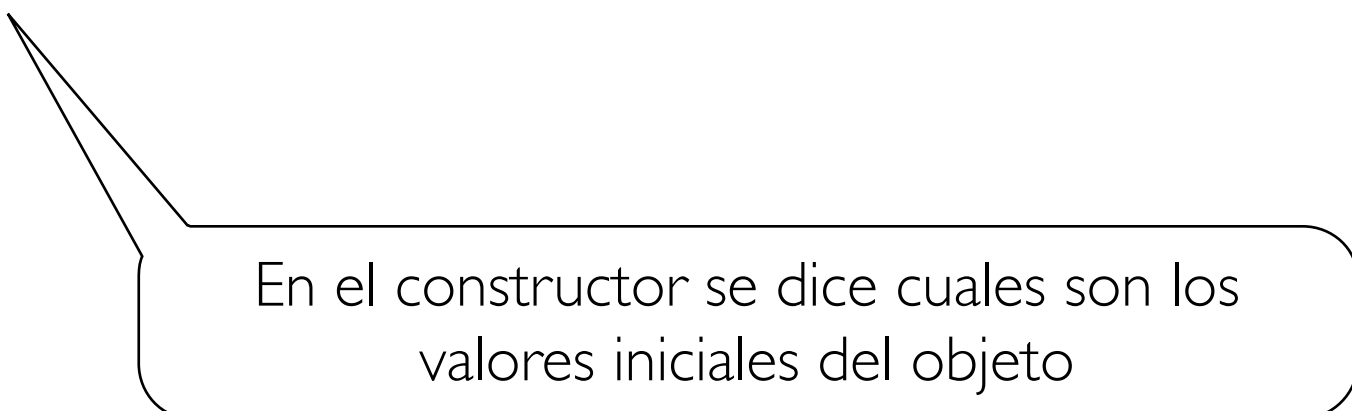
Los clientes invocan a los constructores explícitamente:

```
c1: Circulo  
...
```

```
c1:= nuevo Circulo();  
...
```



Se crea un nuevo circulo



En el constructor se dice cuales son los valores iniciales del objeto

PROGRAMAS OO

- Un programa orientado a objetos es una colección de clases.
- Hay una clase principal (generalmente llamada main), que es donde se comienza la ejecución.
- Hay solo una instancia de la clase principal!

clase Principal

clase principal

metodos

principal [publico]

c1,c2: Circulo;

...

c1:= nuevo Circulo();

c1.nuevo_radio(10);

...

fin

código del programa
utilizando otras clases

JAVA

- JAVA es un lenguaje orientado a objetos.
- Tipos básicos: *int*, *double*, *float*, *char*, *boolean*, etc. (Los tipos básicos no son clases!).
- Clases, referencias, herencia, polimorfismo.
- Gran variedad de librerías definidas.
- *Garbage collector*.
- Portabilidad de código objeto.

Un Ejemplo Simple

Clase Pair

```
/**
 * Pair: A simple example of class in JAVA.
 * @author Pablo Castro
 */
```

```
public class Pair{
    private int x;
    private int y;
```

Datos privados
solo visibles dentro
de la clase

```
/**
 * Constructor of the class
 */
```

```
public Pair(){
    x = 0;
    y = 0;
}
```

Constructor

```
/**
 * Constructor of the class
 * @param x the first component
 * @param y the second component
 * @precondition true
 * @postcondition creates the new object
 */
```

```
public Pair(int x, int y){
    this.x = x;
    this.y = y;
}
```

Otro constructor, con
parámetros

```
/**
 * Get the first coordinate
 * @return the first value of the pair
 * @precondition true
 * @postcondition return the first item
 */
```

```
public int fst(){
    return this.x;
}
```

```
/**
 * Get the second coordinate
 * @return the second value of the pair
 * @precondition true
 * @postcondition return the second value.
 */
public int snd(){
    return this.y;
}
```

```
/**
 * Change the first coordinate
 * @param x change the first value
 * @precondition true
 * @postcondition change x
 */
public void changeFst(int x){
    this.x = x;
}
```

```
/**
 * Change the second coordinate
 * @param y change the second value
 * @precondition true
 * @postcondition change the second value
 */
public void changeSnd(int y){
    this.y = y;
}
```

```
}
```


Ejemplo de Main

visibilidad publica:
cualquier clase la
puede usar

Clase cuadrado,

```
/**
 * Un simple ejemplo de utilizacion de Pair.java
 * @author Pablo Castro
 */
public class Cuadrado{

    public static void main(String[] args){

        if (args.length != 1)
            System.out.println("*** Uso: Cuadrado <integer>");
        else{
            int input = Integer.parseInt(args[0]);
            int res = input * input;
            System.out.println("El resultado es: "+res);
        } // fin de else

    } // fin de main

} // fin de clase
```

Método main, la
clase con el método
main es principal

Ejemplo II

Método que computa el
máximo y mínimo de un arreglo
forma iterativa

```
/*
 * @param i   the starting of the array
 * @param j   the end of the array
 * @result A pair with the min and max
 * @precondition 0 <= i < a.length & 0 <= j < a.length
 * @postcondition (min, max), in the case of empty array (0,0)
 */
public static Pair minMaxArray(int[] a, int i, int j){
    if (i>j){
        Pair result = new Pair(0,0);
        return result;
    }
    else{
        Pair res_sub = new Pair(a[i], a[i]);
        int k = i;
        while (k <= j){
            if (a[k] < res_sub.fst())
                res_sub.changeFst(a[k]);
            if (a[k] > res_sub.snd())
                res_sub.changeSnd(a[k]);
            k++;
        }
        return res_sub;
    }
} // end minMaxArray
```

Ejemplo II

Se debe comentar bien!

Método que computa el máximo y mínimo de forma recursiva

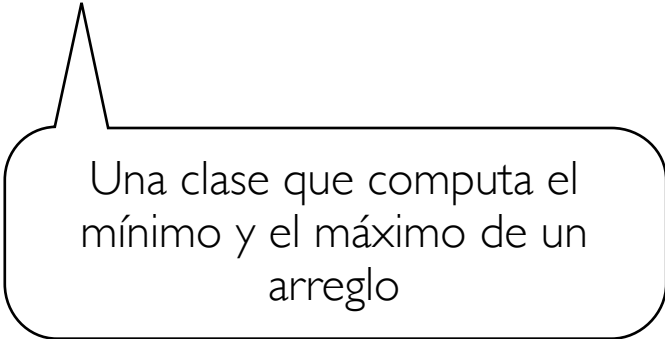
```
/*
 * @param i the starting of the array
 * @param j the end of the array
 * @result A pair with the min and max
 * @precondition 0 <= i < a.length & 0 <= j < a.length
 * @postcondition (min, max), in the case of empty array (0,0)
 */
public static Pair rMinMaxArray(int[] a, int i, int j){
    if (i>j){
        Pair result = new Pair(0,0);
        return result;
    }
    else{
        if (i == j) // caso base
            return new Pair(a[i], a[i]);
        else{ // caso recursivo
            Pair res_sub = rMinMaxArray(a, i+1, j);
            int min = Math.min(res_sub.fst(), a[i]);
            int max = Math.max(res_sub.snd(), a[i]);
            Pair result = new Pair(min, max);
            return result;
        }
    }
}
} // end rMinMaxArray
```

Ejemplo II

```
/**
 * Una clase simple que usa Pair.java para calcular el minimo y maximo de un arreglo
 * @author Pablo Castro
 */

public class MinMax{

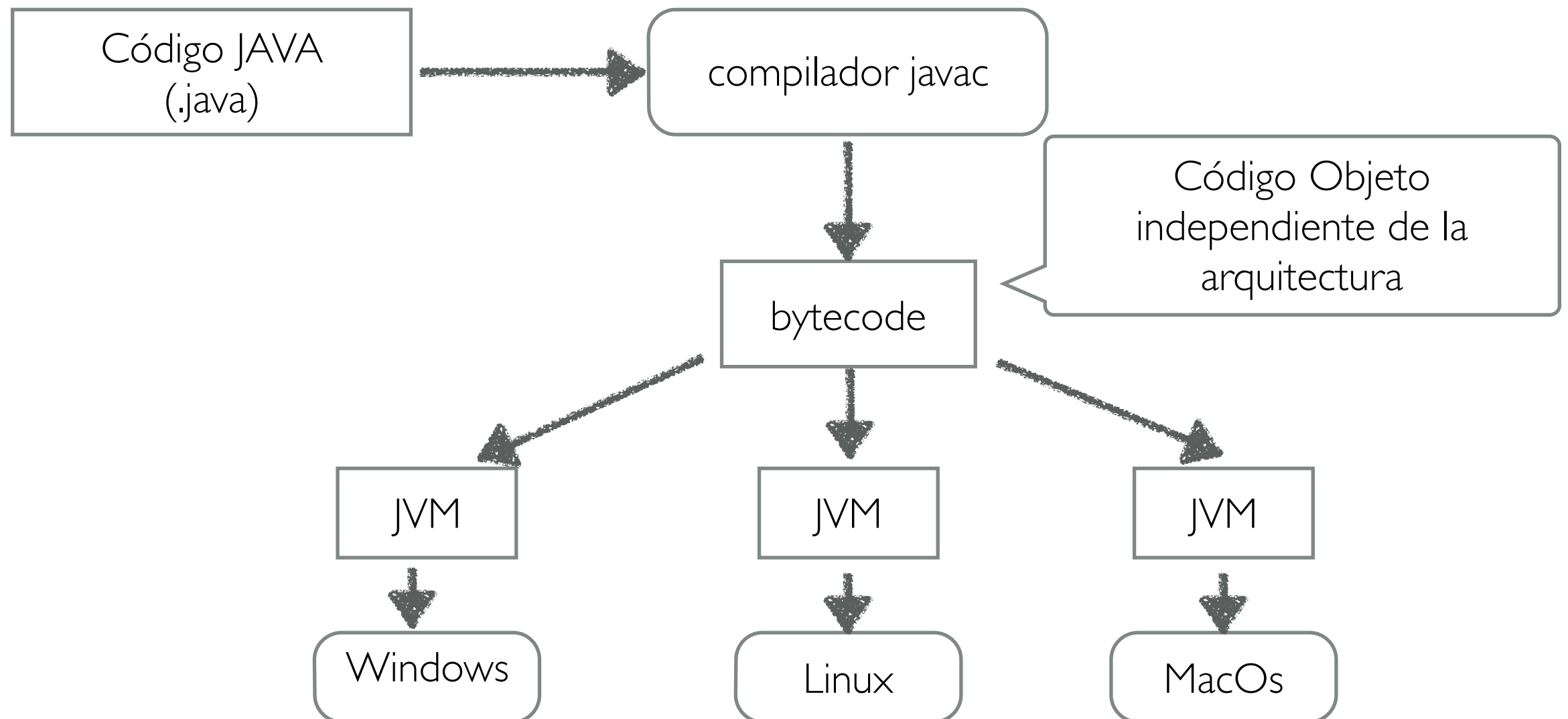
    public static void main(String[] args){
        int[] sample1 = {4,7,3,6,8,1,9,2};
        System.out.println("Result of iterative MaxMin");
        Pair result = minMaxArray(sample1, 0, sample1.length-1);
        System.out.println("Min: "+ result.fst() + " Max: "+ result.snd());
        System.out.println("Result of recursive MaxMin");
        Pair rresult = rMinMaxArray(sample1, 0, sample1.length-1);
        System.out.println("Min: "+ rresult.fst() + " Max: "+ rresult.snd());
    }
}
```



Una clase que computa el
mínimo y el máximo de un
arreglo

LA MAQUINA VIRTUAL DE JAVA (JVM)

Como el nombre lo dice, es una implementación virtual de un computadora:



JVM

PC, registros para
operaciones, etc.

Registros

Stack

Heap

Method
Area

Argumentos de los
métodos, variables
locales, etc

Parte en donde se
guardan los objetos

Código de los métodos

